Design of High-Throughput Inter-PE Communication with Application-Level Flow Control Protocol for Many-Core Architectures

Jyu-Yuan Lai
Dept. of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 30013
jylai@mx.nthu.edu.tw

Po-Yu Chen
Dept. of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 30013
sunshi0583@gmail.com

Yu-Hsun Chen
Dept. of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 30013
adidas90035@hotmail.com

Ting-Shuo Hsu
Dept. of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan 30013
tingshuo.hsu@gmail.com

Chih-Tsun Huang
Dept. of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 30013
cthuang@cs.nthu.edu.tw

Jing-Jia Liou
Dept. of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan 30013
jjliou@ee.nthu.edu.tw

ABSTRACT

With current trend of increasing the number processing elements (PEs) on a single chip, on-chip network provides a fast and reliable interconnect technology for highly parallel applications. Yet, the end-to-end data throughput at software layer on a NoC (Network-on-Chip) platform often cannot match the hardware native speed without an efficient hardware/software interface. In this paper, we present a high-throughput PE-to-PE communication unit with a corresponding driver layer on NoC-based many-core architectures. The proposed communication unit with application-level flow control can handle complicated inter-PE communication for practical parallel applications. The maximum throughput of a unidirectional transmission with flow control protocol at application-level is 2687.3 Mbps (normalized at operating frequency of 100MHz), where the native NoC speed is 3200 Mbps. As a comparison, a software-based protocol is only rated at 148.5 Mbps. The communication unit is also area-efficient at only 19.2K gates, which is roughly 3.2% of a single in-order RISC-based PE.

Categories and Subject Descriptors

C.5.4 [**Computer Systems Organization**]: Computer System Implementation—*VLSI Systems*

General Terms

Design

Keywords

Inter-PE communication, application-level flow control protocol, many-core architecture

MES '13 June 23 - 24 2013, Tel-Aviv, Israel

1. INTRODUCTION

As the demand of high-performance and longer battery-endurance for ubiquitous applications keeps exploding, traditional optimization techniques, such as increasing operating frequency straightforwardly, are no longer favorable for modern resource-constrained systems. Many-core System-on-Chip (SoC) has recently been considered an attractive design style to boost the throughput with much more energy-efficient operation. Moreover, for the flexibility and the scalability, NoC-based interconnection is widely adopted for the implementation of many-core systems.

Many NoC-based many-core processors have been published [1, 2, 3, 4, 5, 6, 7, 8, 9]. The processor pipeline in the Raw microprocessor in [1, 2] is directly integrated with the programmable switch to enable single-cycle data movement operation. In [3], the Asynchronous Array of simple Processors (AsAP) also intends to minimize delay of nearest-neighbor interprocessor communication. And, the architecture is extended to implement a heterogeneous 167-processor computational platform in [4]. In [5, 6], special registers are utilized in the 80-tile processor to boost the frequency and bandwidth of on-chip network traffic. The 48-core processor in [7, 8] utilizes 384 KB of on-chip shared memory with flag allocation for communication between cores.

In summary, most of conventional NoC-based many-core approaches addressed their hardware-driven optimization on inter-PE communication. However, the communication between different PEs becomes much more complicated for modern practical parallel applications. It is hard to predict behavior and sequence of inter-PE communication at runtime. Flow control in network physical layer only ensures that the packets can arrive the destination. The user program has to know the source of coming data and to control the sequence of communication at application-level to avoid system fail.

Therefore, in this paper, we present a high-throughput PE-to-PE communication unit with the application-level flow control protocol for NoC-based many-core architecture. A prototype platform that consists of sixteen PEs with a 4-by-4 mesh-based network is developed [9]. Realistic applications can be compiled using standard GNU C compiler (i.e., *gcc*)

and its tool chain. Note that the applications can be executed on our platform directly without an Operating System (OS) to eliminate the overhead introduced by OS.

The proposed PE-to-PE unit is utilized via developed C-based primitive functions through memory-mapped I/O technique. Thus, complicated inter-PE communication in high-level applications can be achieved easily. To the best of our knowledge, none of the publications mentioned about such a inter-PE communication unit with application-level flow control protocol for practical applications on a demonstrable many-core platform.

The entire many-core platform has been synthesized by using 0.13-μm CMOS technology. The area of our PE-to-PE unit is 19.2K gates, which is only about 3.2% of the PE. Using our PE-to-PE unit, the maximum throughput on a single direction link at application-level is about 2.7 Gbps at the frequency of 100MHz. The experimental results and comparison justify that our inter-PE communication can provide high-throughput with application-level flow control for realistic parallel programs on many-core architectures.

2. PROPOSED MANY-CORE PLATFORM

Fig. 1 shows the prototype platform of our NoC-based many-core system in [9]. The platform consists of sixteen PEs, 4-by-4 mesh-based network request/response switches and links, and external memory. For the flexibility, we adopt the standardized Open Core Protocol (OCP) [10] interface for the interconnection between the PEs and the switches. The architecture also can be scalable by taking the advantage of NoC-based interconnection. In addition, realistic software programs, such as JPEG encoding and object tracking, can be executed on the proposed many-core platform.

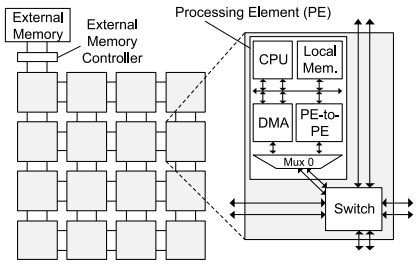

Figure 1: The proposed NoC-based many-core platform.

The PE in our platform comprises a processor core, a local memory, a DMA unit, a PE-to-PE unit, and a local system bus that connects the processor and other peripherals. We adopt the OpenRISC CPU [11] for the open source architecture and convenient tool chain. Our platform is distributed memory architecture. The local memory contains both program instructions and data. The inter-PE communication, which is similar to traditional MPI, is achieved by using the PE-to-PE unit. The DMA is utilized to move bulk data between the local memory and the external memory to improve the efficiency of the processor.

2.1 Proposed PE-to-PE Communication Unit

Fig. 2 (a) shows the block diagram of the proposed PE-to-PE communication unit. There are four types of FIFOs in the PE-to-PE unit. The PE-to-PE engine manipulates the FIFOs based on received commands from the processor. Once the process in sender PE intends to send data to the process in receiver PE, the data is not put into the remote local memory of the receiver PE directly. In contrast, the transmission is done by the PE-to-PE units in both sender and receiver PEs. The PE-to-PE unit is configured by using C-based primitive functions through memory-mapped I/O technique. The details of the communication functions will be shown in Sec. 3. In addition, the OCP interfaces are used to connect the network switch.

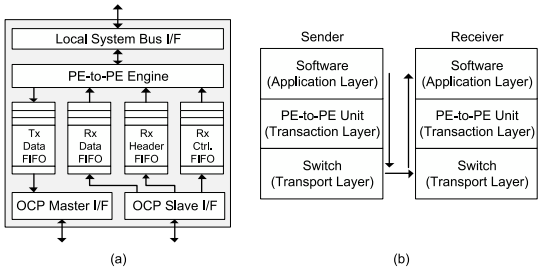

Figure 2: (a) The proposed PE-to-PE communication unit and (b) the network layers in our platform.

The communication network in our many-core platform consists of three layers, i.e., application, transaction, and transport layers, as shown in Fig. 2 (b). We develop on-chip communication library with flow control protocol at application-level for the complicated communication in modern parallel applications.

3. THE ON-CHIP COMMUNICATION

Our on-chip communication library, which is similar to blocking mode of MPI, is developed in C-based functions that can be easily utilized in high-level applications.

3.1 Raw Inter-PE Communication

The *push*() and *pull*() are the most primitive pair of functions for inter-PE communication. The sending process delivers the data by using *push*(), and the receiving process obtains the data by using *pull*(), respectively. The processor core adopts memory-mapped I/O technique to access the PE-to-PE unit. In *push*(), the process specifies the ID of the receiver PE, the address of the sending data in the local memory, the transaction length, and the ID of destination Rx FIFO (i.e., header, control, or data) in the PE-to-PE unit of receiver. Besides, the receiving process assigns target address in the local memory for the coming data by using *pull*().

After the PE-to-PE unit is configured, the data will be transferred by the PE-to-PE units in the sender and receiver directly. Each word (32 bits) can be transferred in 1.2 cycles, which is limited by the burst length (64 words) of native network interconnection. Compared with the software-based approach (the processor moves a word between the local memory and the PE-to-PE unit in 21 cycles), this improved PE-to-PE unit is about 18 times faster.

However, the *push*() and *pull*() do not provide header information at application-level. As a result, when a process needs to obtain data from processes on different PEs concurrently, the receiving process can neither distinguish the source of the incoming data nor ensure the sequence of the sending process. Furthermore, when a sender delivers data to different receivers, the sender cannot tell whether any of the transmission is failed and needs to be re-transmitted.

3.2 Inter-PE Communication with Flow Control Protocol at Application-Level

For the complicated inter-PE communication in various parallel programs, we developed application-level flow control protocol by integrating the primitive *push*() and *pull*().

3.2.1 Single-Mode

The single-word inter-PE communication , i.e., *put*() and *get*() in Fig. 3, is usually performed for synchronization between PEs. The function *put*() embeds ID of the sender PE in a 32-bit header word and delivers it with a 32-bit data word successively to the NoC by using *push*(). The NoC passes the two-word data packet to the destination according to ID of receiver (ID_DestPE). On the other hand, the process on the receiver PE launches *get*() to check the header with specified parameters, e.g., ID of the sender (ID_SrCPE). If the header is correct, the data will be fetched from the Rx header FIFO to the local memory. Also, the acknowledgement ACK will be sent back to the sender, otherwise, $NACK$ denoting failure will be replied. The sender will exit *put*() if ACK is received, otherwise re-transmission will be continued.

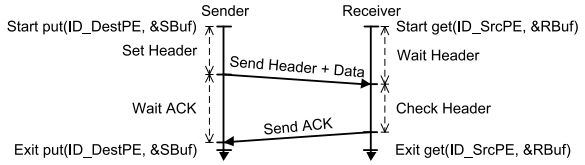

Figure 3: The flow control protocol of put()/get().

3.2.2 Burst-Mode

As shown in Fig. 4, the burst-mode inter-PE communication is similar to the single-mode transmission. The major difference is that the sending process begins with passing only the header packet. The header packet of *send*() comprises both the ID of the sender and the number of words of the following data array. After acknowledgement, the sending process delivers the data array successively. Otherwise, it keeps sending the header packet to the receiver.

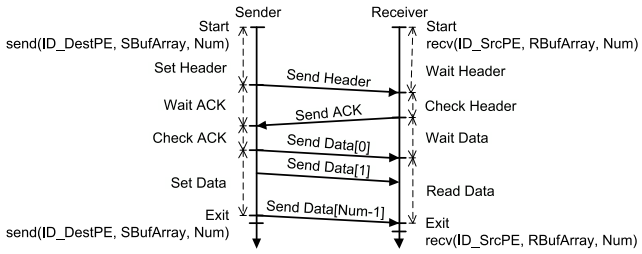

Figure 4: The flow control protocol of send()/recv().

3.3 Discussion on Inter-PE Communication

With the application-level flow control protocol, the user program can tell the source of the coming data by checking the header word. The sequence of communication can also be managed in high-level programs by using this mechanism. In addition, the sender can tell whether a transmission is accomplished by verifying the acknowledgement. Therefore, the PE-to-PE unit with application-level flow control protocol can be used to fulfil complicated inter-PE communication in practical parallel applications.

4. EXPERIMENTAL RESULTS AND COMPARISON

The proposed NoC-based many-core platform has been synthesized using 0.13-μm CMOS technology with the clock rate of 100MHz. The critical path is located in the OpenRISC processor core, and it can be optimized in our future work. The 16 PEs dominate the circuit size, where the mesh network is only 2.04% of area, as shown in Table 1.

Table 1: The hardware area of the proposed many-core platform.

Component	Area (Gates)	Ratio
16 Processing Elements	9,600K	97.93%
4×4 Mesh Network	200K	2.04%
External Memory Controller	3.16K	0.03%
Total	9,803.16K	100%

For further analysis, Table 2 lists the area of each component in a single PE. Storage elements, such as instruction cache, FIFOs of PE-to-PE communication unit, and the local memory, consumes about 88.23% of circuit size of the PE. Note that, in current stage, we emphasize the feasibility and the design exploration of the many-core platform. The size of the local memory can be reduced by using more sophisticated memory hierarchy in our future work. Besides, Table 2 indicates that the area overhead of the PE-to-PE unit is only 3.2% compared with the PE. The implementation result indicates that our PE-to-PE communication unit can fulfil inter-PE data transmission for parallel applications on many-core architectures with negligible hardware overhead.

Table 2: The hardware area of the proposed PE.

Component		Area (Gates)	Ratio
OpenRISC CPU	Arithmetic	25.27K	4.22%
	Instruction Cache (8KB)	63.18K	10.54%
	Others	27.25K	4.55%
	Subtotal	115.70K	19.31%
PE-to-PE Unit	Engine	3.01K	0.50%
	FIFOs	16.19K	2.70%
	Subtotal	19.20K	**3.20%**
DMA		7.52K	1.25%
Local Memory (96KB)		449.55K	74.99%
Others		7.50K	1.25%
Total		599.47K	100%

In Table 3, we compare the throughput of inter-PE communication in our many-core platform by using different on-chip communication functions. The throughput is calculated at application-level by sending 4MB data (1M words) on a single direction link at 100MHz. For the comparison, we also implemented the software-based inter-PE communication. In software-based scheme, data is moved between the local memory and the FIFOs of the PE-to-PE communication unit by the processor itself. The processor demands 21 cycles to move a 32-bit word from the local memory to the FIFOs of the PE-to-PE unit in high-level applications, and vice versa. In addition, software-based scheme costs hundreds of cycles to prepare the header packet. For example, the software-based *put*() takes 351 cycles to set up the header and data compound packet for each transmission, and the software-based *send*() requires 436 cycles to prepare the header packet, respectively. Note that the *send*() transmits

successive data with only one header packet. Consequently, the software-based inter-PE communication can only achieve at most 148.5 Mbps of throughput at application-level.

On the other hand, the throughput of improved PE-to-PE unit is about 10 to 22 times more than that of software-based schemes by optimizing the HW/SW interface. The improved PE-to-PE unit moves data between the local memory and the FIFOs of the PE-to-PE unit directly to relieve the microprocessor from heavy memory routines. The number of cycles for transferring a 32-bit word is reduced from 21 cycles to 1.2 cycles. Note that the throughput of *put*() and *get*() is less than other communication functions, because they are optimized for one-word transmission. In addition, the throughput of inter-PE communication in our platform remains almost the same magnitude regardless of the size of transmitted data.

Table 3: The throughput of inter-PE communication at application-level.

	Software-Based Scheme* (A)	Dedicated Inter-PE Comm. Unit* (B)	Speed Up (B/A)
push() & pull()	148.5 Mbps	2687.6 Mbps	18.4
put() & get()	8.8 Mbps	93.9 Mbps	10.7
send() & recv()	119.1 Mbps	2687.3 Mbps	22.6

*Measured at operating frequency of 100 MHz.

Table 4 compares different many-core designs for inter-PE communication. Note that most of the previous works did not report the throughput of inter-PE communication at application-level. Our PE-to-PE communication unit with the proposed flow control protocol can accomplish complicated data transmission between different PEs for realistic high-level applications on many-core architectures.

Table 4: Comparison for inter-PE communication.

	Inter-PE Comm.	Mechanism for Complicated Inter-PE Comm.	Application-Level Throughput
Ours	**SW-Based**	**No**	**148.5 Mbps**
		Flow Control Protocol	**119.1 Mbps**
	HW-Based	**No**	**2687.6 Mbps**
		Flow Control Protocol	**2687.3 Mbps**
[1, 2]	SW-Based	n.a.	n.a.
[3]	SW-Based	n.a.	n.a.
[4]	SW-Based	n.a.	n.a.
[5, 6]	RIB[1]	n.a.	n.a.
[7, 8]	MPB[2]	Flag Allocation	104.9 Mbps[3], * 1638.4 Mbps[4], *

[1]Router Interface Buffer (RIB); [2]Message Passing Buffer (MPB); [3]The size of data is less than 16 KB; [4]The size of data is larger or equal to 16 KB; *Normalized to 100MHz.

The on-chip interconnection of the many-core architectures in [1, 2, 3, 4] is optimized for nearest-neighbor interprocessor communication with register-mapped transmission. This mechanism fits specific applications, e.g., stream processing. In addition, the mechanism is hard to control the complicated inter-PE communication at application-level, e.g., managing multiple dynamic data transmission which crossing variable distance. The 80-tile processor in [5] focuses on physical network transmission. The applications were implemented in assembly code and optimized manually [6], which is hard to be utilized for the complicated inter-PE communication at high-level applications. The 48-core processor in [7] adopts flags to enforce a safe order of neighborhood communication. However, this mechanism is also difficult to ensure the behavior of concurrent multiple-to-one transmission which is a typical communication pattern in realistic applications. Because the receiver cannot determine which core set the flag. In addition, when the size of transmitted data is larger than 16 KB (the size of MPB for each two cores), the throughput of inter-PE communication at application-level in [7] will degrade significantly, i.e., about 104.86 Mbps normalized to 100MHz. Our PE-to-PE communication unit outperforms the message-passing in [7] in terms of throughput for massive data transmission between PEs. Besides, compared with the size of MPB in [7], i.e., 384 KB of shared memory, our PE-to-PE communication consuming 19.2K gates in each core is fairly area-efficient.

5. CONCLUSIONS

We have presented a high-throughput PE-to-PE communication unit that can fulfil complicated data transmission between PEs at application-level for practical user programs on many-core architectures. On-chip communication library for the PE-to-PE communication unit is proposed to facilitate the development of practical parallel applications. By using 0.13-μm CMOS technology, the area of the PE-to-PE unit is only 3.2% of the PE, i.e., 19.2K gates. The maximum throughput of inter-PE communication with flow control protocol at application-level can achieve 2687.3 Mbps which is 22.6 times more than software-based schemes. The implementation results and comparison show that our inter-PE communication unit is effective for complicated and high-throughput on-chip communication on many-core architectures.

6. ACKNOWLEDGMENTS

This work was supported in part by National Science Council, Taiwan, under Contract NSC 100-2628-E-007-009 and NSC 101-2220-E-007-026 and by Industrial Technology Research Institute of Taiwan, ROC.

7. REFERENCES

[1] E. Waingold *et al.* Baring it all to software: Raw machines. *IEEE Computer*, 30(9):86-93, Spet. 1997.

[2] M. B. Taylor *et al.* The Raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2): 25-35, Mar. 2002.

[3] B. Baas *et al.* AsAP: A fine-grained many-core platform for DSP applications. *IEEE Micro*, 27(2):34-45, Mar. 2007.

[4] D. N. Truong *et al.* A 167-processor computational platform in 65 nm CMOS. *IEEE Journal of Solid-State Circuits*, 44(4): 1130-1144, Apr. 2009.

[5] S. R. Vangal *et al.* An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1): 29-41, Jan. 2008.

[6] T. G. Mattson *et al.* Programming the Intel 80-core network-on-a-chip Terascale processor. In *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1-11, Nov. 2008.

[7] J. Howard *et al.* A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling. *IEEE Journal of Solid-State Circuits*, 46(1):173-183, Jan. 2011.

[8] T. G. Mattson *et al.* The 48-core SCC processor: the programmer's view. In *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1-11, Nov. 2011.

[9] J.-Y. Lai *et al.* Design and analysis of a many-core processor architecture for multimedia applications. In *Proc. Asia-Pacific Signal and Information Processing Association (APSIPA) Annual Submit & Conference*, pages 1-6, Dec. 2012.

[10] OCP-IP. *Open Core Protocol Release 2.2*. OCP-IP Association, Redwood City, Jan. 2007

[11] OpenCores. OpenRISC 1000 architecture manual. http://opencores.org/or1k/Main_Page, June 2011.